useContext() or Redux

Redux is used to manage state of our entire application...

useContext hook in react allows us to consume props within the component tree.

Without having to manually passing down to intermediate components without PROP drilling

useContext() or Redux

1. Scalability & Predictability.
2. Centralized State Management.
3. Middleware for Side Effects.
4. Debugging Tool.
5. Separation of Concerns.
6. Consistency and Ecosystem.
7. Immutability & Pure Functions.

**Scalability & Predictability.**

Redux: Designed for managing global state in larger, more complex applications.

Context API: Best suited for small to medium-sized applications or for passing state deeply through the component tree.

**Centralized State Management.**

Redux: Centralizes all the application state in a single store.

Context API: Best suited for small to medium-sized application or for passing state deeply through the component tree.

**Middleware for Side Effects.**

Redux: Middleware like Redux Thunk or Redux Saga allows you to handle side effects (eg., async operations like API calls) in a more structured and scalable manner. This makes it easier to manage complex asynchronous logic.

Context API: Does not have built-in middleware or support for advanced side-effect handling. You would need to manually integrate side effects into your logic, which can become complicated as the app grows.

**Debugging Tool.**

Redux: Comes with the Redux DevTools that provide powerful debugging capabilities like tracking state changes, time travel, and inspecting dispatched actions.

Context API: Lacks built-in developer tools, making debugging and tracking changes more difficult, especially when the state is complex or deeply nested.

**Separation of Concerns.**

Redux: Encourages a clear separation between actions (what happens), reducers (how state changes), and the UI (how state is updated). This separation can make larger apps more maintainable.

Context API: Tends to combine the state and the logic within the same provider component, which can lead to less clear separation, making the app harder to maintain as it grows.

**Consistency and Ecosystem.**

Redux: Has a vast ecosystem of tools, middleware, and extensions (e.g., Redux Toolkit) that make complex state management task easier and more standardized.

Context API: Dose not have such an ecosystem and lacks these advanced integrations.

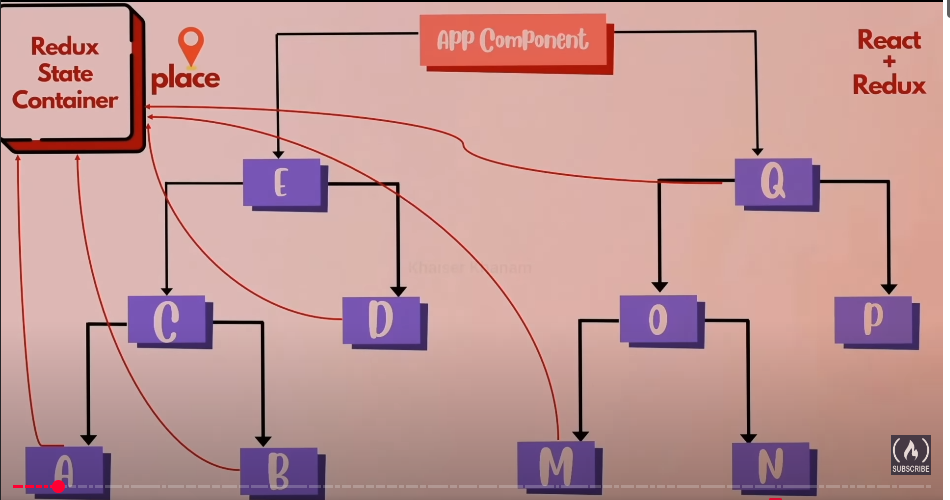
**Immutability & Pure Functions.**

Redux: Enforces a strict immutable state structure through reducers, which helps to prevent unintended mutations and makes the state transitions more predictable.

Context API: You manage immutability and pure functions on your own, which can lead to more mistakes in larger apps if not done carefully.

Should Redux be added to all your application?

When should I use Redux in my React Application?



Redux is better suited for large, complex applications that require structured and centralized state management, side effects handling, and optimization.

Context API is simpler and works well for small to medium-sized applications where state management is relatively straightforward and doesn’t require sophisticated tools.

To make package.json file,used [ npm init –yes ]

**3 Core concepts of REDUX**

A **store** that keeps your app’s data.

An **action** that explains how the data changes.

A **reducer** that makes the changes to the data based on the actions.

**3 Essential Principles**

1. Single Source of Truth
2. State is Read-Only
3. Changes are Made with Pure Functions.

**Single Source of Truth**

First principle single source of truth the entire application state is store in a single object,

Object is nothing but which is having properties and values track and manage state changes consistently across the entire application and where we are going to manage our entire state we will managing inside our redux store.

In simple terms,

Maintain our entire application sate in as single object which would be managed by the redux store.

**State is Read-Only**

The only way to change the state is by dispatching actions.

Actions are plain objects that describe what happened, ensuring that state changes are predictable and traceable.

Dispatch

Action  
type:’increment’,  
payload:5

Store  
Current State

The third principle covers

How should the state transformed

In simple terms,

To update the state of your app, we need to let Redux know about that with an action.

**Changes are Made with Pure Functions.**

To specify how the state changes in response to actions, we write pure functions called reducers.

Dispatch

Action  
type: ‘increment’,  
payload:5

Store  
Reducers  
Reducers  
Current State

In simple terms,

It tells we need to write pure reducers to determine how the state changes. (By dispatching actions) and reducer will be changing the state so reducers are pure functions which takes the current state and action as its arguments and returns a new state without mutating our existing state

Reducer=(previousState, action)=>newState

What we understood we are having a Redux store in which entire applications state is store and changes can only be happening to the state only when we dispatch action and how the changes are made to the state by writing our Pure Functions that is nothing but a read users so reducers function previous state that is numbers of base which was thousand and action order type and based on what the action type is a new state object is returned from red user.

Const reducer=(state,action)=>{

Switch(action,type){

Case’ORDER\_PIZZA’:

Return{

pizzaBase:state.pizzaBase-1

}

}

}

